Recognition of Ordered Tree-Shellable Functions Based on OBDDs (Algorithm Engineering as a New Paradigm)

AUTHOR(S):

Takenaga, Yasuhiko

CITATION:

Takenaga, Yasuhiko. Recognition of Ordered Tree-Shellable Functions Based on OBDDs (Algorithm Engineering as a New Paradigm). 数理解析研究所講究録 2001, 1185: 191-198

ISSUE DATE:

2001-01

URL:

http://hdl.handle.net/2433/64624

数理解析研究所講究録
1185巻 2001年 191-198

Recognition of Ordered Tree-Shellable Functions Based on OBDDs

Yasuhiko TAKENAGA

Department of Computer Science, The University of Electro-Communications
Chofu, Tokyo 182-8585, Japan
takenaga@cs.uec.ac.jp

Abstract: In this paper, we consider the complexity of recognizing ordered tree-shellable Boolean functions when Boolean functions are given as OBDDs. An ordered tree-shellable function is a positive Boolean function such that the number of prime implicants equals the number of paths from the root node to a 1-node in its ordered binary decision tree representation. We show that given an OBDD, it is possible to check within polynomial time if the function is ordered tree-shellable with respect to the variable ordering of the OBDD.

Keywords: tree-shellable Boolean function, OBDD, prime implicant, shellability

1 Introduction

It is important to clarify the properties of Boolean functions in various fields of computer science. Prime implicant is a very important concept on the theory of Boolean functions. Every positive Boolean function is uniquely represented by an irredundant DNF.

A tree-shellable function is a positive Boolean function defined by the relation between its prime implicants and binary decision tree (BDT) representation: there exists a BDT representation such that the number of prime implicants equals the number of paths from the root to a leaf labeled 1 in the BDT [20]. An ordered tree-shellable function is a special case of a tree-shellable function such that the BDT must be an ordered BDT. In this paper, we deal with the complexity of recognizing ordered tree-shellable functions.

An ordered tree-shellable function is a kind of shellable function. Shellable Boolean functions play an important role in many fields. The notion of shellability was originally used in the theory of simplicial complexes and polytopes (for example, in [9, 12]). More recently, it is studied for its importance on reliability theory (for example, in [2, 3, 19]).

If a shellable function f is given with the order of terms to make it shellable, it has the following good properties. First, one can easily solve the following problem.

[Union of Product Problem] ([3])
Input: $Pr[x_i = 1](1 \leq i \leq n)$, $f(x_1, ..., x_n)$
Output: $Pr[f(x_1, ..., x_n) = 1]$

$Pr[A]$ represents the probability of event A. This is the problem of computing the reliability of some kind of systems. Each variable represents the state of a subsystem. A subsystem is operative if the variable has value 1. If a Boolean function f is shellable, one can easily compute the exact value of $Pr[f = 1]$ using the orthogonal DNF representation of f.

Second, it is easy to compute the dual of the function. The dual of a Boolean function $f(x_1, ..., x_n)$ is defined by $f^d = \overline{f(\overline{x_1}, ..., \overline{x_n})}$. Thus, if the BDT representation of a Boolean function f is given, it is possible to compute the BDT representation of f^d only by exchanging a 1-edge and a 0-edge for every variable node and exchanging label 1 and label 0 for every leaf node. It is not known if the DNF representation of the dual f^d can be computed from the DNF representation of f in time polynomial to the input and output size. So the problem is still interested in by many researches (for example, in [5, 13, 15]). The classes of Boolean functions which can be dualized in polynomial time include 2-monotonic functions (which include threshold functions) [11, 8, 18], aligned functions [6], positive k-DNFs [14], matroid functions [17] etc.

Ordered tree-shellability has been studied un-

der the name of lexico-exchange property in [3, 7, 19]. It is proved in [20] that these two properties are equivalent. This property is interested in because a function with this property is a shellable function such that the function is shellable by the lexicographic order of terms. In addition to the above properties of shellable functions, as described in the definition, an ordered tree-shellable function has an intersting relation between its ordered BDT representation and prime implicants.

In this paper, we consider the complexity of checking if a given function is ordered tree-shellable or not. When a Boolean function is given as its DNF representation, it is known to be NP-complete [7]. The complexity of this problem depends on the maximum number m of literals in a term. For $m \leq 2$, it is polynomial time computable [4] and the class of ordered tree-shellable functions and that of shellable functions coincide. In this case, the order of terms is also found in polynomial time. If we also give a variable ordering π as a part of the input, it is possible to check if the function is ordered tree-shellable with respect to π within polynomial time.

In this paper, we consider the case when a Boolean function is given as its Ordered Binary Decision Diagram (OBDD) representation. An OBDD [1, 10] is a directed acyclic graph that represents a Boolean function. As OBDDs are widely used in many applications due to their good properties, it is worth considering the case when an OBDD is given as an input of recognition problems [16]. If a function is given as an OBDD, Union of Product Problem can be solved in polynomial time and dualization can also be executed in a similar way as the case of BDTs. However, this problem is still worth consideration due to the relation between an OBDD representation of a function and its prime implicants. This problem is interesting because it is considered difficult to find the prime implicants of the function given as an OBDD in general.

We show that it is possible to check if the function is ordered tree-shellable with respect to the variable ordering of the given OBDD within polynomial time. It should be noted that the result does not follow from the similar result for the case when a function is given in DNF. When the variable ordering is fixed, it is easily seen that the class of functions represented by polynomial size OBDDs and that represented by polynomial size DNFs are incomparable.

This paper is organized as follows. In sections 2 and 3, we define basic concepts and graph representations of a Boolean function. In section 4, we give the definition of an ordered tree-shellable function and show its basic properties. In section 5, we study the complexity of checking ordered tree-shellability of a function given as an OBDD. Conclusions and future works are noted in section 6.

2 Basic Definitions

Let $B = \{0,1\}$, n be a natural number, and $[n] = \{1,2,...,n\}$. Especially, $[0] = \emptyset$. Let π be a permutation on $[n]$. π represents a total order of integers in $[n]$.

Let $f(x_1,...,x_n)$ be a Boolean function. We denote $f \geq g$ if $f(x) = 1$ for any assignment $x \in \{0,1\}^n$ which makes $g(x) = 1$. An *implicant* of f is a product term $\bigwedge_{i\in I} x_i \bigwedge_{j\in J} \overline{x_j}$ which satisfies $\bigwedge_{i\in I} x_i \bigwedge_{j\in J} \overline{x_j} \leq f$, where $I, J \subseteq [n]$. An implicant which satisfies $\bigwedge_{i\in I-\{s\}} x_i \bigwedge_{j\in J} \overline{x_j} \not\leq f$ for any $s \in I$ and $\bigwedge_{i\in I} x_i \bigwedge_{j\in J-\{t\}} \overline{x_j} \not\leq f$ for any $t \in J$ is called a *prime implicant* of f.

An expression of the form $f = \bigvee_{k=1}^{m} \left(\bigwedge_{i\in I_k} x_i \bigwedge_{j\in J_k} \overline{x_j} \right)$ is called a *disjunctive normal form Boolean formula* (DNF), where $I_k, J_k \subseteq [n]$ and $I_k \cap J_k = \emptyset$ for $k = 1,...,m$. A *positive* DNF (PDNF) is a DNF such that $J_k = \emptyset$ for all k. If f can be represented as a PDNF, it is called a positive Boolean function. For simplicity, we call that I_k is an implicant or a term of a positive function. A PDNF is called *irredundant* if $I_k \subseteq I_l$ is not satisfied for any k,l $(1 \leq k,l \leq m, k \neq l)$. For an irredundant PDNF, let $PI(f)$ be the set of all I_k. $PI(f)$ represents the prime implicants of f. In the following of this paper, we consider only positive functions and we assume that a function is given as an irredundant PDNF $f = \bigvee_{k=1}^{m} \bigwedge_{i\in I_k} x_i$.

3 Graph Representations of Boolean Functions

3.1 Binary Decision Tree

A *Binary Decision Tree* (BDT) is a labeled tree that represents a Boolean function. A leaf node of a BDT is labeled by 0 or 1 and called a value node. Any other node is labeled by a variable and called a variable node. Let $label(v)$ be the label of node v. Each node except leaf nodes has two outgoing edges, which are called a *0-edge* and a *1-edge*. Let $edge_0(v), edge_1(v)$ denote the nodes pointed to by the 0-edge and the 1-edge of node v respectively. The value of the function is given by traversing from the root node to a leaf node. At a node, one of the outgoing edges is selected according to the value of the variable. The value of the function is 0 if the label of the leaf is 0, and 1 if the label is 1.

A path from the root node to a leaf node labeled 1 is called a *1-path*. On every 1-path, each variable appears at most once. A path P of a BDT is represented by a sequence of literals. If the k-th edge on a 1-path P is the 1-edge (0-edge, resp.) from the node labeled by x_i, positive literal x_i (negative literal $\overline{x_i}$, resp.) is the k-th element of P. For simplicity, we denote $\tilde{x_i} \in P$ when $\tilde{x_i}$ is included in the sequence representing P, where $\tilde{x_i}$ is either x_i or $\overline{x_i}$. Let $pos(P_k)$ ($neg(P_k)$, resp.) be the set of indices of variables whose positive (negative, resp.) literals are in P_k.

When the 0-edge and the 1-edge of node v point to the nodes representing the same function, v is called to be a *redundant node*. A BDT which has no redundant node is called a *reduced BDT*. In the following of this paper, a BDT means a reduced BDT.

A BDT is called an *ordered* BDT (OBDT) if there is a total order of variables which is consistent with the order that variables appear on any path from the root to a leaf. The total order of variables for an OBDT is called the *variable ordering*. If $label(v)$ is the k-th element of the variable ordering, we say that k is the level of v and denote $level(v) = k$. Let the level of value nodes be $n + 1$.

The Boolean function that is represented by node v, denoted by f_v, is defined as follows by Shannon's expansion :

$$f_v = \begin{cases} label(v) & \text{(if } v \text{ is a value node)} \\ label(v) \cdot f_{edge_1(v)} + \overline{label(v)} \cdot f_{edge_0(v)} & \\ & \text{(otherwise).} \end{cases}$$

An OBDT represents the function represented by the source.

3.2 Ordered Binary Decision Diagram

An *Ordered Binary Decision Diagram* (OBDD) [1, 10] is a directed acyclic graph that represents a Boolean function. Intuitively, an OBDD is obtained by merging the nodes of an OBDT that represent the same function into a single node. The nodes of an OBDD consist of *variable nodes* and two *value nodes*. One of the variable nodes is the source and the value nodes are sinks. Most of the terminologies defined for OBDTs can be defined similarly for OBDDs.

When two nodes i and j have the same label and represent the same function, they are called *equivalent nodes*. When $edge_1(i) = edge_0(i)$, node i is called a *redundant node*. An OBDD which has no equivalent nodes and no redundant nodes is called a *reduced* OBDD. It is known that a Boolean function is uniquely represented by a reduced OBDD, provided that the variable ordering is fixed. In the following of this paper, we assume w.l.o.g. that an OBDD means a reduced OBDD. The size of an OBDD is the total number of nodes.

4 Ordered Tree-Shellable Boolean Function

We first give the definition of tree-shellable and ordered tree-shellable functions.

Definition : A positive Boolean function f is *tree-shellable* when it can be represented by a BDT with exactly $|PI(f)|$ 1-paths.

If a BDT T represents $f = \bigvee_{k=1}^{m} \bigwedge_{i \in I_k} x_i$ and has exactly m paths, we say that T witnesses that f is tree-shellable.

Definition : A positive Boolean function f is *ordered tree-shellable with respect to* π if it can be represented by an OBDT with variable ordering π which has exactly $|PI(f)|$ 1-paths. f is *ordered*

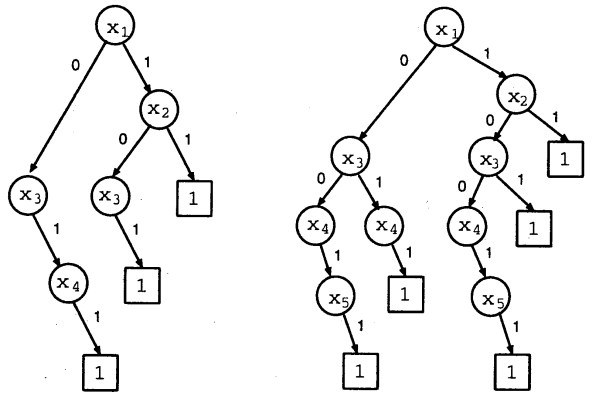

Figure 1: An example of an ordered tree-shellable function.

tree-shellable if there exists π such that f is ordered tree-shellable with respect to π. We call π to be the *shelling variable ordering* of f.

Proposition 1 [20] *If* $f = \bigvee_{k=1}^{m} \bigwedge_{i \in I_k} x_i$ *is tree-shellable, there exists a BDT* T *representing* f *which satisfies the following conditions.*

- T *has* m *1-paths* $P_1, ..., P_m$.
- *Each* P_k *corresponds to a term* I_k *by the rule that* $i \in I_k$ *iff* $x_i \in P_k$.

Figure 1 (a) is an example of an ordered tree-shellable function. Note that the leaf nodes with label 0 and the edges that point to them are omitted in this figure. f is ordered tree-shellable with respect to variable ordering $x_1\ x_2\ x_3\ x_4$. Figure 1 (b) is an example of a function which is not ordered tree-shellable. With variable ordering $x_1\ x_2\ x_3\ x_4\ x_5$, the BDT representing g has five 1-paths, which does not equal the number of prime implicants. It is not difficult to check that g is not ordered tree-shellable with respect to any other variable ordering.

The following proposition shows that tree-shellability is a kind of shellability as its name shows.

Definition : Let f be a positive Boolean function represented by a PDNF $f = \bigvee_{k=1}^{m} \bigwedge_{i \in I_k} x_i$ and π_T be an order of terms of f. f is *shellable* with respect to π_T if there exist $J_1, ..., J_m (\subseteq [n])$ which satisfy the following conditions.

1. For any l $(1 \leq l \leq m)$,
$$\bigvee_{k=1}^{l} \bigwedge_{i \in I_{\pi_T(k)}} x_i = \bigvee_{k=1}^{l} \Big(\bigwedge_{i \in I_{\pi_T(k)}} x_i \bigwedge_{j \in J_{\pi_T(k)}} \overline{x_j} \Big).$$
2. For any s, t such that $1 \leq s < t \leq m$, $(I_s \cap J_t) \cup (I_t \cap J_s) \neq \emptyset$.

f is shellable if there exists π_T such that f is shellable with respect to π_T. π_T is called the *term order* of f. For tree-shellable functions, the term order is determined from the BDT that witnesses that f is tree-shellable. If a 1-edge is always a right edge as in Figure 1, the term order is obtained by ordering the 1-paths in a BDT from the right one to the left one.

Proposition 2 [20] *Let a positive Boolean function* $f = \bigvee_{k=1}^{m} \bigwedge_{i \in I_k} x_i$ *be tree-shellable with respect to* π_T, *and* P_k *be the 1-path of the BDT representing* f *that corresponds to* I_k. *Then*
$$\bigvee_{k=1}^{l} \bigwedge_{i \in I_{\pi_T(k)}} x_i = \bigvee_{k=1}^{l} \Big(\bigwedge_{i \in pos(P_{\pi_T(k)})} x_i \bigwedge_{j \in neg(P_{\pi_T(k)})} \overline{x_j} \Big)$$
for any l $(1 \leq l \leq m)$.

This proposition is obvious from properties of a BDT.

As an ordered tree-shellable function is tree-shellable, Proposition 1 and 2 also hold for ordered tree-shellable functions.

The next corollary is clear from the proof of Theorem 4 of [20].

Corollary 1 *Let* T *be an OBDT with variable ordering* π *that represents a Boolean function* f. f *is ordered tree-shellable with respect to* π *iff there exists* I_t *which satisfies* $I_t \subsetneq pos(P_i) \cup \{l\}$ *and* $l \in I_t$ *for any 1-path* P_i *of* T *and any* $\overline{x_l} \in P_i$,

5 Checking Ordered Tree-Shellability Based on OBDDs

In this section, we consider the complexity of checking ordered tree-shellability of a Boolean

function given as an OBDD. We consider only the ordered tree-shellability with respect to the variable ordering π of the given OBDD. If a positive Boolean function is given in DNF, it is possible to check whether it is ordered tree-shellable with respect to given π within polynomial time. However, it does not imply that the similar result holds when a Boolean function is given as its OBDD representation.

Theorem 1 *Given an OBDD with variable ordering π, it is possible to check if the Boolean function represented by the OBDD is ordered tree-shellable with respect to π or not within polynomial time.*

We should note that if there exist several variable orderings which are consistent with the given OBDD, either all of them are the shelling variable ordering or none of them is the shelling variable ordering.

We prove Theorem 1 in the rest of this section. We first give the polynomial time algorithm to check ordered tree-shellability. Let $lev(u,v) = \min\{level(u), level(v)\}$. In this algorithm, 0 represents the value node labeled 0.

[Algorithm CheckOTS]

1. Check if the OBDD represents a positive function. If not, it is not ordered tree-shellable. Else, $A_i = \emptyset$ for all i $(2 \leq i \leq n+1)$.

2. For $i = 1$ to n, repeat (a) and (b).

(a) For any node v in level i do:
if $edge_0(v) \neq 0$
$A_{lev(edge_0(v),edge_1(v))}$
$= A_{lev(edge_0(v),edge_1(v))} \cup \{(edge_0(v), edge_1(v))\}$

(b) For any pair $(u,v) \in A_i$ do:
if $level(u) > i$
$A_{lev(u,edge_0(v))}$
$= A_{lev(u,edge_0(v))} \cup \{(u, edge_0(v))\}$
else if $level(v) > i$
if $edge_0(u) \neq 0$
$A_{lev(edge_0(u),v)}$
$= A_{lev(edge_0(u),v)} \cup \{(edge_0(u), v)\}$
else do:
$A_{lev(edge_1(u),edge_1(v))}$
$= A_{(edge_1(u),edge_1(v))} \cup \{(edge_1(u), edge_1(v))\}$
if $edge_0(u) \neq 0$
$A_{lev(edge_0(u),edge_0(v))}$
$= A_{(edge_0(u),edge_0(v))} \cup \{(edge_0(u), edge_0(v))\}$

3. The given OBDD represents an ordered tree-shellable function iff no pair of the form (u,u) is generated in step 2.

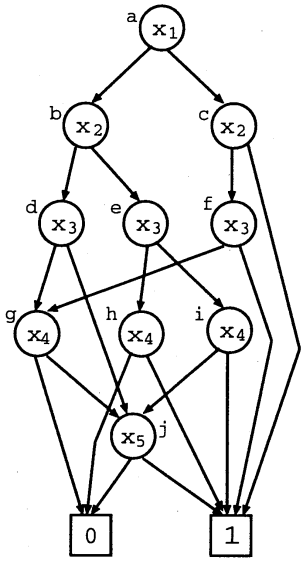

Figure 2: An example of input for Algorithm CheckOTS.

In this algorithm, A_i $(2 \leq i \leq n+1)$ is a set of pairs of nodes.

Here we give an example of how this algorithm works. The OBDD in Figure 2 is an example of the input. In this figure, the left edge from each node is a 0-edge and the right one is a 1-edge. The alphabet beside each variable node is the name of the node. We can check that the OBDD represents a positive Boolean function. In fact, it represents $x_1x_2 + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_5$. In step2, for $i = 1$, a pair (b,c) is generated in step2a. Thus, $A_2 = \{(b,c)\}$. For $i = 2$, (d,e) and $(f,1)$ are generated in step2a, and (d,f) and $(e,1)$ are generated from (b,c) in step2b. Thus, $A_3 = \{(d,e),(d,f),(e,1),(f,1)\}$. Similarly, after $i = 3$, $A_4 = \{(g,g),(g,h),(g,j),(g,1),(h,i),(h,1),(j,i)\}$ and $A_5 = \{(j,1)\}$. After $i = 4$, $A_5 = \{(j,j),(j,1)\}$ and $A_6 = \{(1,1)\}$. After $i = 5$, $A_6 = \{(1,1)\}$. Here, we find three pairs of the form (u,u). Therefore, the function is not ordered tree-shellable with respect to variable ordering $x_1\ x_2\ x_3\ x_4\ x_5$.

We consider the time complexity of the above algorithm. Let m be the size of the given OBDD. As shown in [16], step 1 can be executed in m^2 time. In step 2a, throughout n iterations, each variable node appear exactly once. Thus, it takes $O(m)$ time. In step 2b, throughout n iterations, the same pair may be generated many times. However, as the number of different generated pairs of nodes is less than m^2, the total number of generated pairs is less than $2m^2$. Thus, Algorithm CheckOTS runs in $O(m^2)$ time.

Now we should prove that Algorithm CheckOTS correctly checks the ordered tree-shellability of the given Boolean function. This proof consists of two stages. We first show in Lemma 1 that there exists a pair of 1-paths P_i, P_j that satisfies some condition iff the function is not ordered tree-shellable with respect to the variable ordering. Then we show in Lemma 2 that the algorithm correctly detects such pair of 1-paths.

We give some basic facts on a Boolean function f and the OBDD representing f. Note that f may not be ordered tree-shellable. We call a 1-path P_j which satisfies $pos(P_j) = I_i$ the *main path* of I_i. If P_j is a main path of some prime implicant, we call P_j a main path. If an OBDT T witnesses that f is ordered tree-shellable, any 1-path of T is a main path. We call a 1-path P_j which satisfy $I_i \subseteq pos(P_j)$ a *corresponding path* of I_i.

Proposition 3 *1. For any prime implicant I_i, there exists a main path of I_i.*
2. Any path is a corresponding path of some prime implicant.

Proof 1. As I_i is a prime implicant, there exists no 1-path P_s which satisfies $pos(P_s) \subsetneq I_i$. To make the value of the function 1 for an assignment such that $x = 1$ iff $x \in I_i$, there must be a 1-path P_s which satisfies $([n] \setminus I_i) \cap pos(P_s) = \emptyset$. To satisfy both of them, there must be a 1-path P_s satisfying $pos(P_s) = I_i$.
2. A 1-path P_j makes the value of the function 1 for the assignment such that $x = 1$ iff $x \in P_j$. However, it should be 0 if there is no prime implicant I_i which satisfies $I_i \subseteq pos(P_j)$. □

From Proposition 3 and the definition of ordered tree-shellable functions, we can see that there exists a pair of 1-paths both of which are corresponding paths of the same prime implicant iff f is not ordered tree-shellable. The next lemma shows that we have only to detect special ones among such pairs of 1-paths. The number of generated pairs is decreased by using this lemma.

Lemma 1 *Let T be an OBDD representing f with variable ordering π. f is not ordered tree-shellable with respect to π iff there exists a pair of 1-paths P_i, P_j in T which satisfies $pos(P_i) \subsetneq pos(P_j)$ and $|pos(P_j) \setminus pos(P_i)| = 1$.*

Proof [if] If there exists a pair of 1-paths P_i, P_j satisfying $pos(P_i) \subsetneq pos(P_j)$, P_i and P_j are corresponding paths of the same prime implicant. That is, at least one of them is not a main path. [only if] Assume f is not ordered tree-shellable. Then from Corollary 1, for some 1-path P_i and $\overline{x_l} \in P_i$, there does not exist I_t that satisfies $I_t \subseteq pos(P_i) \cup \{l\}$ and $I_t \not\subseteq pos(P_i)$. For such P_i and x_l, let P_j be the path traversed by the assignment such that $x_k = 1$ iff $k = l$ or $x_k \in P_i$. If P_i is a corresponding path of $I_{i'}$, P_j is also a corresponding path of $I_{i'}$ because it cannot be a corresponding path of any other prime implicant. Therefore, P_j satisfies $pos(P_j) = pos(P_i) \cup \{l\}$. That is, $pos(P_i) \subsetneq pos(P_j)$ and $|pos(P_j) \setminus pos(P_i)| = 1$ are satisfied. □

In the example of Figure 2, (g, g) detects the existence of paths $P_i = \overline{x_1}\ \overline{x_2}\ \overline{x_3}\ x_4\ x_5$ and $P_j = x_1\ \overline{x_2}\ \overline{x_3}\ x_4\ x_5$ that join at node g.

In the second step, we have to show that Algorithm CheckOTS correctly detects such a pair of paths. In other words, we have to prove the following lemma.

Lemma 2 *Algorithm CheckOTS finds a pair of nodes (u, u) iff there exists a pair of 1-paths P_i, P_j as described in Lemma 1.*

Proof [if] For simplicity, we assume w.l.o.g. that the variable ordering is $x_1\ x_2 \cdots x_n$. Let e_i^s be the first node in P_i that belongs to a level larger than s. Let P_i and P_j first diverge at a node in level t. We prove that for any $s \geq t$, the pair (e_i^s, e_j^s) is generated in the algorithm.

We prove it by induction on s. A pair (e_i^t, e_j^t) is generated in step2a. Because $x_t \in P_j$ and $x_t \notin P_i$ hold, for x_k $(k > t)$, either
i) $x_k \in P_i, x_k \in P_j$,
ii) $\overline{x_k} \in P_i, \overline{x_k} \in P_j$,

iii) $\overline{x_k} \in P_i, x_k \notin P_j, \overline{x_k} \notin P_j$,
iv) $\overline{x_k} \in P_j, x_k \notin P_i, \overline{x_k} \notin P_i$ or
v) $x_k \notin P_i, \overline{x_k} \notin P_i, x_k \notin P_j, \overline{x_k} \notin P_j$
must hold. Note that i) and ii) can occur when both e_i^{k-1} and e_j^{k-1} belong to level k, iii) and iv) occur when either of them belongs to level k, and v) occur when neither of them belongs to level k. Thus, if (e_i^{s-1}, e_j^{s-1}) is generated, we can easily see from the operations in step2b that (e_i^s, e_j^s) is generated in any of the above cases.

If P_i and P_j join at node u in level r, $(u,u) = (e_i^{r-1}, e_j^{r-1})$. Therefore, (u,u) never fails to be generated.

[only if] We prove that for any pair (v,w) generated in the algorithm

($*$) there exist 1-paths P_v and P_w such that P_v is a path from the source to v, P_w is a path from the source to w, $pos(P_v) \subsetneq pos(P_w)$ and $|pos(P_w) \setminus pos(P_v)| = 1$.

If it holds, when there exists a pair (u,u), P_i and P_j of Lemma 1 are obtained by appending a path from u to the value node labeled 1 to both P_v and P_w.

We prove it by induction on the number of iterations in step 2. In the first iteration, one pair is generated in step 2a and the pair satisfies ($*$). We assume that all the pairs generated in the i-th iteration ($i < s$) of step 2 satisfy condition ($*$). In the s-th iteration,
a) any pair generated in step 2a clearly satisfies ($*$), and
b) any pair generated in step 2b from a pair (v', w') satisfies ($*$) by appending literals corresponding to the edges shown in the algorithm to $P_{v'}$ and $P_{w'}$ because (v', w') satisfies ($*$). □

6 Conclusion

In this paper, we have considered the complexity of checking ordered tree-shellability of a Boolean function given as an OBDD. We have shown that given an OBDD, it is possible to check within polynomial time if the function is ordered tree-shellable with respect to the variable ordering of the OBDD. However, it seems difficult to check if the given function is ordered tree-shellable with respect to the other variable orderings.

References

[1] S.B. Akers, "Binary decision diagrams," IEEE Trans. Comput. vol.C-27, no.6, pp.509–516, 1978.

[2] M.O. Ball and G.L. Nemhauser, "Matroids and a reliability analysis problem," Math. Oper. Res. vol.4, pp.132–143, 1979.

[3] M.O. Ball and J.S. Provan, "Disjoint products and efficient computation of reliability," Oper. Res., vol.36, no.5, pp.703–715, 1988.

[4] C. Benzaken, Y. Crama, P. Duchet, P. L. Hammer and F. Maffray, More Characterizations of Triangulated Graphs, Journal of Graph Theory vol.14, pp.413–422, 1990.

[5] J.C. Bioch and T. Ibaraki, "Complexity of identification and dualization of positive boolean functions," Inform. and Comput. vol.123, pp.50–63, 1995.

[6] E. Boros, "Dualization of aligned boolean function," Discrete Applied Math. (to appear), RUTCOR Research Report 9-94, 1994.

[7] E. Boros, Y. Crama, O. Ekin, P.L. Hammer, T. Ibaraki and A. Kogan, "Boolean normal forms, shellability and reliability computations," SIAM J. Discrete Mathematics (to appear), RUTCOR Research Report 3-97, 1997.

[8] E. Boros, P.L. Hammer, T. Ibaraki, and K. Kawakami, "Polynomial time recognition of 2-monotonic positive boolean functions given by an oracle," SIAM J. Computing, vol.26, no.1, pp.93–109, 1997.

[9] H. Brugesser and P. Mani, "Shellable decompositions of cells and spheres," Math. Scand., vol.29, pp.199–205, 1971.

[10] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Comput., vol.C35, no.8, pp.677–691, 1986.

[11] Y. Crama, "Dualization of regular boolean functions," Discrete Applied Math., vol.16, pp.79–85, 1987.

[12] G. Danaraj and V. Klee, "Shellings of spheres and polytopes," Duke Math. J., vol.41, pp.443–451, 1974.

[13] C. Domingo, N. Mishra and L. Pitt, "Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries," Machine Learning, vol.37, pp.89–110, 1999.

[14] T. Eiter and G. Gottlob, "Identifying the minimal transversals of a hypergraph and related problems," SIAM J. Computing, vol.24, no.6, pp.1278–1304, 1995.

[15] M.L. Fredman and L. Khachiyan, "On the complexity of dualization of monotone disjunctive normal forms," J. Algorithms, vol.21, pp.618–628, 1996.

[16] T. Horiyama and T. Ibaraki, "Knowledge-base representation and recognition of positive/horn functions on ordered binary decision diagrams," Technical Report of IEICE, COMP98-85, pp.17–24, 1999.

[17] K. Makino and T. Ibaraki, "The maximum latency and identification of positive boolean functions," SIAM J. Computing, vol.26, pp.1363–1383, 1997.

[18] U.N. Peled and B. Simeone, "Polynomial-time algorithm for regular set-covering and threshold synthesis," Discrete Applied Math., vol.12, pp.57–69, 1985.

[19] J.S. Provan and M.O. Ball, "Efficient recognition of matroids and 2-monotonic systems," in R. Ringeisen and F. Roberts (eds.), Applications of Discrete Mathematics, pp.122–134, SIAM, Philadelphia, 1988.

[20] Y. Takenaga, K. Nakajima and S. Yajima, "Tree-shellability of Boolean functions," Technical Report of IEICE, COMP97-54, pp.71–78, 1997.